Hybrid Tuning of an Evolutionary Algorithm for Sensor Allocation

Myriam Abramson
Naval Research Laboratory
Washington, DC 20375
Myriam.Abramson@nrl.navy.mil

Ian Will
Naval Research Laboratory
Washington, DC 20375
ian.will@nrl.navy.mil

Ranjeev Mittu
Naval Research Laboratory
Washington, DC 20375
ranjeev.mittu@nrl.navy.mil

***Abstract*—The application of evolutionary algorithms to the optimization of sensor allocation given different target configurations requires the tuning of parameters affecting the robustness and run time of the algorithm. In this context, parameter settings in evolutionary algorithms are usually set through empirical testing or rules of thumb that do not always provide optimal results within time constraints. Design of experiments (DOE) is a methodology that provides some principled guidance on parameter settings in a constrained experiment environment but relies itself on a final inductive step for optimization. This paper describes a sensor allocation tool developed for intelligence, surveillance and reconnaissance (ISR) in the maritime domain and introduces a hybrid methodology based on DOE and machine learning techniques that enables the tuning of an embedded particle swarm optimization (PSO) algorithm for different scenarios.**

I. Introduction

A sensor allocation tool (SAT) has been brought up as part of the Interactive Scenario Builder (BUILDER) to automate the optimal placement of sensors for intelligence, surveillance and reconnaissance (ISR). Builder is a 3D tactical decision aid and mission planning system that provides insight into, and visualization of, the radio frequency (RF) environment and is accessible to the DoD community [1]. One of the key capabilities that BUILDER offers is its suite of RF propagation models implemented by another DoD owned software tool called EMPIRE. A user typically creates a scenario consisting of platforms and threats (and their anticipated or projected movements) within the BUILDER environment. When constructing a scenario, the different types of sensors on the various platforms must also be specified as well as their operating characteristics.

Platform allocation optimization is a complex problem that involves (1) the optimization of path planning for the platforms, (2) the configuration of the sensors on a platform, and (3) the coordination of the platforms themselves. Previous work has included team models of unattended ground sensor networks [2] and cooperative path planning of unmanned aerial vehicles (UAVs) [3]. By discretizing the geospatial search space, platform allocation optimization can be solved as a combinatorial optimization problem similar to role allocation problems where the tasks are waypoints on a grid. A *signal graph* within BUILDER discretizes the geospatial space and stores the RF propagation from the predicted ISR targets at each node taking into account terrain, no-fly zones, and other constraints such as weather. ISR targets may be physical objects (target for radars, infrared or optical), electronic transmissions (targets for passive electronic sensors), or areas of interest, varying in position and altitude. Knowing the current location of those ISR targets is a wicked problem involving the successful tracking of those targets and therefore implying the optimization of the platform allocation itself. Here, the current and predicted motion of the ISR targets in a scenario is inferred from track pattern history. Figure 1 illustrates the signal graph as an overlay to the problem space. Finding a possible allocation of platforms is then a matter of checking whether a sensor on a platform can "receive" a target signal at this location. The size of this signal graph (i.e. the number of nodes) is manually set but ultimately should depend on the density of the platforms and the targets and could affect the run-time of the optimization algorithm. Using the nodes of the signal graph, the A* algorithm, native to BUILDER, computes the feasibility of a path from the location of the platform to its destination. This evaluation is included in the optimization of the platforms location in the sensor allocation tool in order to minimize distance cost. The planning horizon, determined by the length of the scenario, is the time at which the allocation of sensors is optimized although the period can be discretized to continuously optimize against a dynamic situation. The time step discretization should therefore be a function of the change in situational awareness. Figure 2 describes the architecture of SAT within BUILDER.

While intelligence analysts are trained to manipulate the scenario parameters, it is desirable that the optimization parameters automatically adjust to the scenario without the human-in-the-loop. This paper is a step in this direction. The remainder of this paper is organized as follows. Section II describes the particle swarm optimization used by the sensor allocation tool. Section III summarizes previous work in parameter tuning and introduces our hybrid approach while Section IV presents experimental results followed by conclusions in Section V.

II. Optimization

Several optimization algorithms apply to the sensor allocation problem. Evolutionary computation is particularly

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUN 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **Hybrid Tuning of an Evolutionary Algorithm for Sensor Allocation**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,4555 Overlook Avenue SW,Washington,DC,20375**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES **2011 IEEE Conference on Evolutionary Computation, 5-8 June, New Orleans, LA.**		

14. ABSTRACT

The application of evolutionary algorithms to the optimization of sensor allocation given different target configurations requires the tuning of parameters affecting the robustness and run time of the algorithm. In this context, parameter settings in evolutionary algorithms are usually set through empirical testing or rules of thumb that do not always provide optimal results within time constraints. Design of experiments (DOE) is a methodology that provides some principled guidance on parameter settings in a constrained experiment environment but relies itself on a final inductive step for optimization. This paper describes a sensor allocation tool developed for intelligence surveillance and reconnaissance (ISR) in the maritime domain and introduces a hybrid methodology based on DOE and machine learning techniques that enables the tuning of an embedded particle swarm optimization (PSO) algorithm for different scenarios.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**7**	

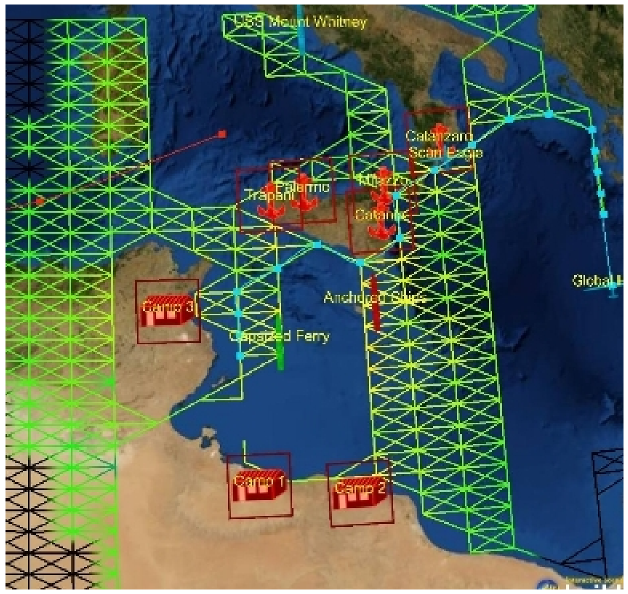


Figure 1. Signal Graph

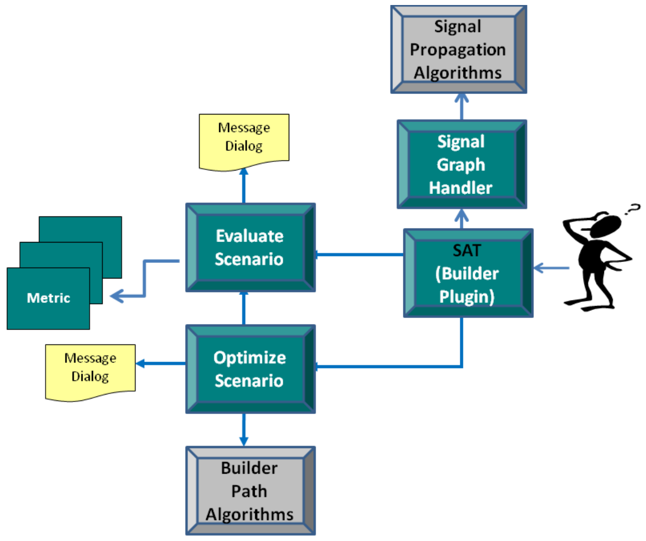


Figure 2. SAT architecture as a BUILDER plugin

attractive because of the ease with which to combine objective functions (i.e. metrics) through multi-objective optimization and its capability to address non-linear classes of optimization problem. Evolutionary computation can also refine other optimized solutions (for example, dynamic programming solutions have also been applied to this problem) to achieve some global solution metric or to replan in dynamic environments without recomputing an entire solution.

Particle swarm optimization (PSO) is an evolutionary computation algorithm (Alg. 1) based on the social learning metaphor [4] and the reinforcement of past success whereby an agent, represented by an n-dimensional velocity vector, adapts its solution from the solution of one of its neighbors and past performance. It is characterized by its simplicity and

Algorithm 1 Basic PSO algorithm

```
Initialize weight vectors $x$,
  parameters $vmax, w, \varphi_1, \varphi_2$
DO
  $p_g \leftarrow \arg\max_i f(p_i)$ % global best particle
  FOREACH particle $i$
     $p_l \leftarrow \arg\max_j f(p_j)$ % local best particle
     FOREACH dimension $d$
       $r_1 \leftarrow random(0,1)$
       $r_2 \leftarrow random(0,1)$
       $v_{id} \leftarrow wv_{id} + \varphi_1 r_1 (p_{ld} - x_{id}) + \varphi_2 r_2 (p_{gd} - x_{id})$
       $v_{id} \leftarrow sign(v_{id}) min(abs(v_{id}), vmax)$
       $x_{id} \leftarrow x_{id} + v_{id}$
     ENDFOR
  decrement $w$
  ENDFOR
UNTIL termination criterion
```

efficiency due to its small population size and number of parameters. The search through the problem space, controlled by the n-dimensional velocity vector, gives the learning agent a "particle" movement characteristic. In summary, three types of interaction are usually distinguished: (1) a top-down type of interaction based on normative knowledge of the "global best", *gbest*; (2) a bottom-up type of interaction based on internal personal performance, *pbest* (in addition, *pbest* acts as the agent's episodic memory of past performances); and (3) a neighborhood knowledge of the "local best", *lbest*. The cognitive (*pbest*) and social influences (*gbest* or *lbest*) are modulated by the stochastic parameters φ_1 and φ_2 respectively. The shape of neighborhood models determines the choice of *gbest* or *lbest* and affects the convergence of a swarm [5]. An inertia parameter w, decreasing with time, acts as the momentum in neural networks in controlling the exploration-exploitation trade-off of the search. The velocities v are bounded to a value $\pm vmax$ to control the search. As agents interact, subsets of the population become more similar and successful together. The topology of the population determines the local best neighbor and not the similarity of the vectors themselves as is found in instance-based learning.

In addition to N, the population size, and C, the maximum number of iterations (cycles), the parameters of the PSO algorithm, $vmax, w, \varphi_1, \varphi_2$, are usually set to default values $(\pm\infty, [0.9 - 0.7], 1.49, 1.49)$ that have been shown to work empirically [4]. It has also been shown that the parameters must satisfy the relationship $1 > w > \frac{1}{2}(\varphi_1 + \phi_2) - 1 > 0$ for convergence [6]. The number of iterations specified serves also as an upper bound for other parameters detecting convergence (for example, the number of consecutive iterations with minimum diversity radius in the population) but that capability is hard to achieve. The topology of the swarm can also affect results. A small neighborhood size l slowly increasing with time in proportion to the number of iterations can have slow convergence but has better chances not to get stuck in a local minima by exploring the search space more thoroughly [7], [6]. Assuming that φ_1 and φ_2 divides equally and assuming the full range of w, the parameters left to optimize for PSO are N, C, and φ, the sum of φ_1 and φ_2, where $2 < \varphi < 4$ to satisfy

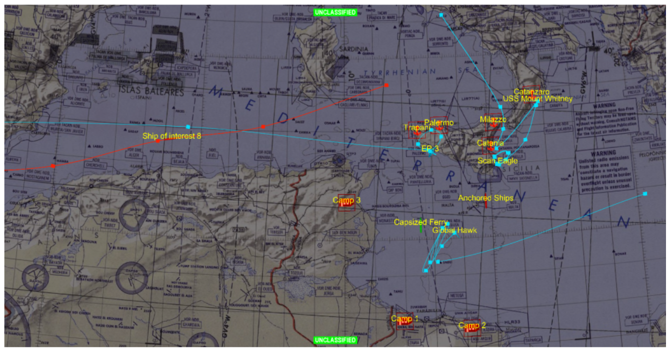
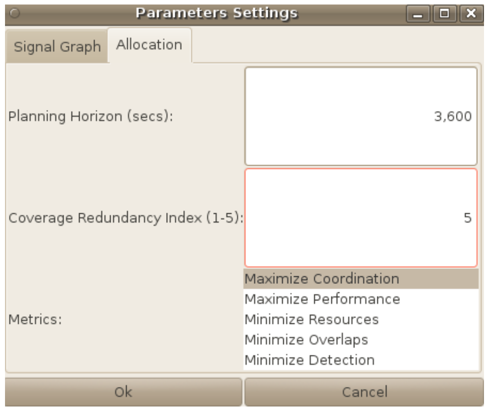


Figure 3. Red/Blue platform motions visualization in BUILDER


Parameters Settings

Signal Graph | Allocation

Planning Horizon (secs): 3,600

Coverage Redundancy Index (1-5): 5

Metrics: Maximize Coordination, Maximize Performance, Minimize Resources, Minimize Overlaps, Minimize Detection

Ok | Cancel

Figure 4. Interface for optimization settings in SAT. The coverage redundancy index indicates whether a penalty should incur due to overlaps in target coverage.

the convergence relationship with w. In addition, the size of the signal graph R (asuming equal dimensions for rows and columns) is another parameter specific to the sensor allocation tool. The size of the search space is the Cartesian product of the dimensions of each parameter $N \times C \times \varphi \times R$ with appropriately discretized ranges. The n-dimensional vector to be optimized is a list of candidate destinations for each friendly platform (asset) with varying range $[-r, r]$ where r is the number of nodes in the signal graph. A negative value for r indicates to ignore this platform in the sensor allocation problem. For example, [-41,33,25,-5] indicates to allocate platform 2 to grid point 33 and platform 3 to gridpoint 25 and to ignore platform 1 and 4. Algorithm 2 describes the sensor allocation optimization routine where the update of the particle position follows Alg. 1. The platform motions are automatically updated and visualized within BUILDER from the best solution obtained (Fig. 3).

Multiple PSO populations optimized with different fitness functions can combine through the velocity vector update equation in Alg. 1 [8]. The selection of a population to influence another can either be made randomly or through a topological arrangement of the populations (e.g. ring topology). The "best" solution is then the solution on the Pareto frontier of the different fitness functions. We have decomposed a coordination quality metric designed for multi-agent systems [9] into its different components – resources, performance and failures – to offer some flexibility through multi-objective optimization in achieving desired outcomes. Figure 4 illustrates the interface for the optimization settings while the PSO parameters proper and the size of the signal graph are specified through an auxiliary input file hidden from the user. Providing automated guidance on those latter parameters given a scenario is needed for the robustness of the optimization in an operational environment.

Algorithm 2 Sensor Allocation Optimization Loop
Random restart reintializes a particle that fails to meet the solution constraints.

```
INPUTS: assets, targets, signal graph,
        fitness metric
OUTPUT: Best platform assignments and paths
Initialize population vectors
ITERATE
  FOREACH candidate solution
     Evaluate fitness of solution:
     Check destination feasibility with A*
     Check for signal received
     Check for time-to-destination constraint
     Update particle position (Alg. 1) or
        random restart
END
```

III. Parameter Tuning

Given a certain configuration of sensors, platforms and targets, an automated parameter tuning capability is necessary for making allocation algorithms robust and usable while preserving optimality. A good survey of tuning methods for evolutionary algorithms can be found in [10] where algorithmic and search approaches are distinguished. The main characteristics of these approaches as they relate to sensor allocation optimization where different scenarios define different fitness spaces are summarized below.

While it is possible to evolve local parameters together with the solution parameters in an evolutionary algorithm, thereby increasing the search space, it is difficult to evolve the meta-level parameters such as the population size and the number of iterations themselves in this manner. Toward this end, another evolutionary algorithm, dimensioned with default values, can search the metal-level parameter space [11]. This meta-evolutionary approach does not however infer directly from the problem space (our scenario) to the search space parameters and needs to be run anew with each new scenario. Estimating probability distributions for the parameters correlated with increased performance provides some robustness to noise and situation changes [12] but still no direct inference from different scenarios is possible.

Search methods such as racing [13] and sharpening [10] manipulate the parameter vectors to be evaluated to incrementally converge to an optimal set of values. Sequential parameter optimization [14] is a search method that evaluates new parameter vectors with a principle approach based on a sampling technique. A regression tree can guide the search for optimal

Run	A	B
1	-1	-1
2	-1	1
3	1	-1
4	1	1
5	0	0
6	0	0
7	$-\sqrt{2}$	0
8	$\sqrt{2}$	0
9	0	$-\sqrt{2}$
10	0	$\sqrt{2}$

Table I
2-FACTOR CCD DESIGN TEMPLATE

Run	A	B	C	D	E
1	-1	-1	-1	-1	1
2	-1	-1	-1	1	-1
3	-1	-1	1	-1	-1
4	-1	-1	1	1	1
5	-1	1	-1	-1	-1
6	-1	1	-1	1	1
7	-1	1	1	-1	1
8	-1	1	1	1	-1
9	1	-1	-1	-1	-1
10	1	-1	-1	1	1
11	1	-1	1	-1	1
12	1	-1	1	1	-1
13	1	1	-1	-1	1
14	1	1	-1	1	-1
15	1	1	1	-1	-1
16	1	1	1	1	1
17	0	0	0	0	0
18	0	0	0	0	0
19	0	0	0	0	0
20	0	0	0	0	0
21	0	0	0	0	0
22	0	0	0	0	0
23	0	0	0	0	0
24	α	0	0	0	0
25	$-\alpha$	0	0	0	0
26	0	α	0	0	0
27	0	$-\alpha$	0	0	0
28	0	0	α	0	0
29	0	0	$-\alpha$	0	0
30	0	0	0	α	0
31	0	0	0	$-\alpha$	0
32	0	0	0	0	α
33	0	0	0	0	$-\alpha$

Table II
5-FACTOR CCD DESIGN TEMPLATE WHERE $-\alpha$ AND α MAP TO THE MINIMUM AND MAXIMUM VALUE RESPECTIVELY AND 0 TO THE MIDPOINT. ± 1 MAPS TO SOME DISPLACEMENT FROM THE MIDPOINT. THE FACTORS MAP TO OUR PARAMETERS N, C, φ, R AND SCENARIO TYPE.

values by progressively dividing the parameter space [15]. Our approach differs from search method techniques in that we obtain optimal parameter values from the interpolation of a minimal set of experimental values and where the interpolation is done with a machine learning technique to take into account the interaction of the parameters. This approach enables the decoupling of the characteristics of the simulation itself from the characteristics of the evolutionary algorithm.

Experimental design provides a principled way of finding parameter values that will produce the desired optimization with a minimum number of experiments. Unlike ablation studies that vary one parameter at a time, this approach takes parameter interactions into account. The parameter levels (values) are standardized and encoded so templates for experimental design can apply to different problems with a simple mapping. There is a linearity assumption between input and output in 2-level designs (designs where only high and low parameter values are considered and encoded as ± 1). When that assumption fails, intermediary levels of the input values are explored as in the central-composite design (CCD) [16] and a quadratic model can then be applied to obtain a model of the input feature relationships with the output response (Table I). Characteristics of the situation to include the conditions under which the algorithm will apply as well as the parameters of the optimization algorithm can be part of the experimental design. This approach can therefore provide a direct inference from the problem space to the search space but provides no guidance for complex problems where we don't have a model bias. In addition, we also want to minimize certain input parameters such as the population size and the number of iterations while maximizing the response but experimental design does not give any guidance on constrained optimization.

Our approach leverages from the experimental values found in the CCD design and classifiers from machine learning to learn the optimization algorithm responses to inputs and parameters. The CCD design is a flexible and efficient "classical" design for capturing the feature interactions of quantitative factors. It consists of a full or fractional design portion augmented by a set of intermediate points. We explore the representation change to multi-dimensional tile coding in this problem. Tile coding (derived from CMAC [17], [18]) discretizes continuous state spaces into overlapping regions with possibly different orientations and resolutions (Fig. 5). Tile coding is a useful technique for dimensionality reduction in large state space. Each tile becomes a binary feature in a binary vector that compactly describes a state. This binary vector can in turn be fed to an associative neural network, such as the perceptron, to learn the association of the optimization algorithm's response to a set of inputs and parameters. This change of representation promotes the generalization of the input space such that points close together in the input space are also close in the output space while points that are distant stay distant and capture non-linear feature interactions with a simple associative algorithm. The degree of generalization corresponds to the number of overlapping tile layers (or tilings). Tile coding has been successfully coupled with reinforcement learning methods to learn the value of state-action pairs in continuous state spaces as a weighted linear combination of binary features [19]. Tile coding is similar to hash coding techniques [20] for approximate evaluation. Tile coding can also be used as a technique for projecting a problem from a finite dimensional space where no linearity between inputs and outputs can be found to a higher dimensional space. Support vector machines [21] generalizes this concept for classification problems with

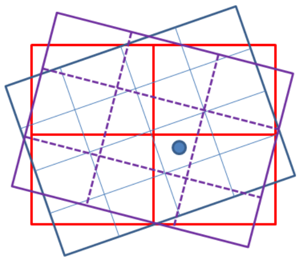


Figure 5. Tile coding: Counting from top-left to bottom-right, the dot is in tile 4 of the first set, tile 5 of the second set, and tile 11 of the third set and can be represented by the feature vector “0001000010000000000000100000”.

a kernel function mapping data points and their dot products to a higher dimensional space.

While tile coding enables the discretization of continuous state spaces, experimental design tests for corner points in the space. To resolve this difference, we tile the space so that the experimental design points are at the center of the tile. Fig. 6 illustrates the one-dimensional mapping of the central composite design points (corner, center and axial) into a 2-layer tile coding with 3 overlapping tiles where $-\alpha$ and $+\alpha$ maps to the minimum and maximum range of the dimension respectively. Each data point in the input space is mapped to two layers and therefore has at least two binary features. In a multidimensional setting, the first tiling has n^p tiles where n is the number of range increments and p is the number of parameters. The second tiling subdivides further the first into n^p tiles leading to a combinatorial explosion of order n^{np}. Our approach is to reduce the multidimensional encoding to a one-dimensional encoding with overlapping tiles and different tile width at each layer. The amount of overlap, tile width, and number of layers affect the generalization quality. Automated approaches to tile generation [22] can find an optimized tile configuration to achieve best prediction performance. The following steps summarize our approach:

1) Collect experiments for the CCD design of the four parameters N,C, φ, and R, in each scenario.
2) Generate tile coding layers (see Alg 3).
3) Convert multi-dimensional encoding of parameters according to their discretization increment from CCD design experiments to one-dimensional encoding.
4) Set binary features by checking one-dimensional encoding tile memberships in each layer to obtain a binary feature vector associated with a response outcome.
5) Train with machine learning methods on binary feature vectors obtained in Step 3.

Then, given a scenario, we'll iterate through the possible parameter values, $N \times C \times \varphi \times R$, to obtain the set of parameter values S given a scenario that will maximize the estimated responses $\hat{y}$ from the trained model obtained in Step 5 while minimizing costs such as population size and cycles (Alg. 4).

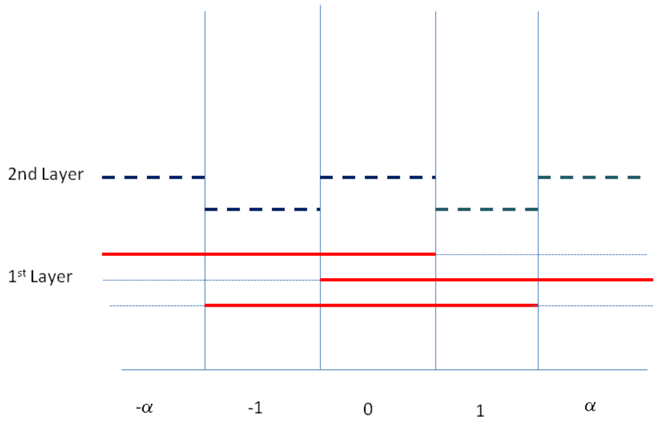


Figure 6. One-dimensional mapping of CCD data points into 2-layer tile coding with 3 overlapping tiles in the first layer

Algorithm 3 Tile generation for a layer given range and tile width with random offset

```
INPUT: range, tileWidth
OUTPUT: tiles
tiles ←{}
overlap ← random($\frac{tileWidth}{2}$)
start ←0
end ←0
WHILE (end < range)
    end ← start+tileWidth
    tiles ← tiles ∪(new Tile(start,end))
    start ← end-overlap
END
```

$$S = \arg\min \max_{N \times C \times \varphi \times R} \hat{y}$$

To scale up to large number of dimensions, a fractional design rather than a full factorial design can be used to gather prototype experiments and a randomized algorithm could replace the iteration through the possible parameter values (Fig. 5). Randomized algorithms provide an approximate solution while bounding the probability of an incorrect solution. Let E_i the event of *not* picking the right optimal value for parameter i. The probability of picking the right optimal value is at most $\frac{1}{v_i}$ for v_i values of parameter i so that $P(E_i) \geq 1 - \frac{1}{v_i}$. Let $M = \max(v_1, ..., v_k)$, the probability of *not* selecting a set of optimal value for k parameters is $P(\cup_{i=1}^{k} E_i) \leq \left(1 - \frac{1}{M}\right)^k$since each parameter value selection is independent. We can repeat the execution of the inner loop N times such that the probability of *not* selecting a set of optimal value is less than a certain threshold δ:

$$N(1 - \frac{1}{M})^k \leq \delta$$

Algorithm 4 Minimize parameter values while maximizing outcome given one fixed parameter

```
NAME: GETBESTPARMS
INPUT: paramIndex, instance, params
OUTPUT: BestOutcome {Instance, response}
BestOutcome ←{instance, 0}
IF (paramIndex == params.length)
  outcome ←classifyInstance(instance)
  RETURN BestOutcome ←{instance, outcome}
END
parameter ←params[paramIndex]
IF fixed parameter

  Outcome ←GETBESTPARMS (paramIndex+1, instance)
  BestOutcome ←
  $\max_{response}$(BestOutcome, Outcome)
END
FOREACH parameter value of parameter
  modify instance to include value
  Outcome ←GETBESTPARMS (paramIndex+1, instance)
  BestOutcome ←
  $\max_{response}$(BestOutcome, Outcome)
END
```

Algorithm 5 Randomized GETBESTPARMS where N minimize the error according to a threshold δ.

```
NAME: GETBESTPARMS
INPUT: paramIndex, instance, params
OUTPUT: BestOutcome {Instance, response}
BestOutcome ←{instance, 0}
IF (paramIndex == params.length)
  outcome ←classifyInstance(instance)
  RETURN BestOutcome ←{instance, outcome}
END
parameter ←params[paramIndex]
IF fixed parameter

  Outcome ←GETBESTPARMS (paramIndex+1, instance)
  BestOutcome ←
  $\max_{response}$(BestOutcome, Outcome)
END
REPEAT N times
  value ←select a random parameter value
  modify instance to include value
  Outcome ←GETBESTPARMS (paramIndex+1, instance)
  BestOutcome ←
  $\max_{response}$(BestOutcome, Outcome)
END
```

IV. Experimental Results

In addition to the four parameters for our PSO optimization (using a global best topology), our scenario variable is represented by the ratio of assets over targets, $\frac{\#assets}{\#targets}$. Finding a continuous representation of the scenario is essential to this problem. Other characteristics of a scenario could include the terrain elevation, atmosphere refraction, particulate count (air clarity), sea-state, wind, and precipitation. Five distinct scenarios were generated to correspond to the five levels of the central composite design (including center and axial points). The range of the variables are described in Table III. The data for a CCD design (including a half fractional design where interaction variables are aliased with a main factor) with 5 factors (Table II) and 16 runs was generated using the sensor allocation tool taking performance, $\frac{\#target\,received}{\#targets}$, as our response variable. Including the CCD points, our experimental sample size was 33 combinations of parameter values of 16 runs each or 528 examples. While the optimization is bounded by the size of the scenario, there are trade-offs between parameter values and response quality. For example, more cycles will not improve the optimization in a large scenario as much as in a smaller scenario. Figure 7 describes the interaction between cycles and scenario size. A linear regression model of the response variable from the CCD design experiments is as follows with a correlation coefficient of 0.88:

Input	Range	Increment
N	10-100	10
C	100-500	100
φ	2.1-3.9	0.3
R	10-50	10
$\frac{\#assets}{\#targets}$	0.23-0.44	given

Table III
Variable ranges and increments for extrapolation

$$response = 0.0005N + 0.0001C - 0.01\varphi + 0.001R + 1.80S + 0.1734$$

In our experiments, only the scenario was found to significantly influence the response in a linear regression model. The resources to maximize the response, i.e. population size, cycles, and graph size, might be unnessary or prohibitive to obtain a good solution within timing constraints. A multi-layer (ML) perceptron (from Weka's toolbench [23]) with default values (3 nodes in the hidden layer) trained with the same experimental data points from the CCD design gives us more flexibility in selecting those parameters using Alg. 4. In addition, using tile coding with 2 layers of 23 and 41 tiles respectively with random offsets (generating 88 binary features), we also trained a multi-layer perceptron with the same dataset converted to tile coding. In both instances, we obtained a correlation coefficient (accuracy) of 0.90 with 10-fold cross-validation. The comparative results presented in Table IV shows that the ML perceptron can give us more flexibility in setting the population size while agreeing with the general results of a linear regression model for the other parameters to achieve maximum performance. There is a significant difference in the results obtained with the ML perceptron in Scenario 1 when compared with default values and tile coding over 10 runs (t-tests p-values less than 0.001). The results also show that the representation change to tile coding with the ML perceptron was not always better than the default values in those experiments but that it has the potential to provide more refinement in setting those parameter values and enlarge the parameter space.

V. Conclusions

We have shown how design of experiments methods can provide guidance on obtaining sample data points from which to train a non-linear classifier to estimate the response given certain parameter values. This capability enables the interactive tuning of an optimization algorithm for a sensor allocation

	Scenario $\frac{\#assets}{\#targets}$	Default Performance (20,200,2.98,20)	ML Perceptron Recommended Settings	ML Perceptron Performance	ML Perceptron Recommended Settings w/ Tile Coding	ML Perceptron w/ Tile Coding Performance
1	0.23	0.59±0.05	(30,500,2.1,50)	0.65±0.0	(40,400,3.6,50)	0.60±0.03
2	0.28	0.68±0.05	(10,500,2.1,50)	0.74±0.06	(40,100,2.7,50)	0.72±0.03
3	0.33	0.79±0.05	(30,500,2.1,50)	0.83±0.03	(10,100,3.9,20)	0.75±0.03
4	0.40	0.92±0.06	(10,500,2.1,50)	0.95±0.05	(10,100,2.1,40)	0.90±0.06
5	0.44	0.89±0.04	(10,500,2.1,50)	0.91±0.03	(10,100,3.9,30)	0.85±0.03

Table IV
COMPARATIVE RESULTS BETWEEN DEFAULTS SETTINGS AND SETTINGS FROM ML PERCEPTRON MODEL (AVERAGED OVER 10 RUNS)

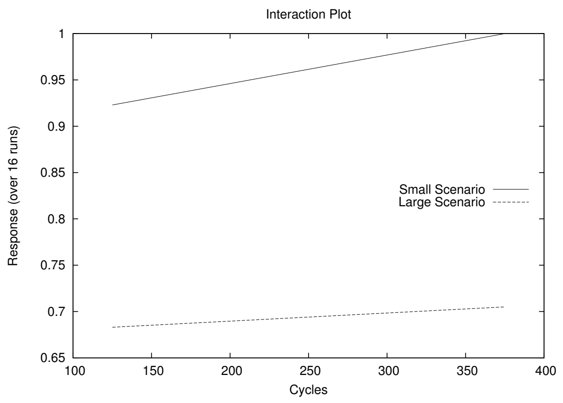


Figure 7. Interaction plot between cycles and scenarios

tool where the complexity of a scenario can be encoded as a parameter itself. More precision in this encoding is needed to capture the different aspects of a situation. The results have shown that the ML perceptron can provide refined guidance on the setting of the parameters for particle swarm optimization while additional encoding of the training set with tile coding was not necessary to obtain better parameter values than default values in those experiments. It was shown however that tile coding can provide more refinement in the settings of the parameter values and additional work would consist of automating tile encoding in conjunction with the standard encoding of experimental design.

The authors would like to acknowledge the Tactical SIGINT Technologies Program Management Office for sponsoring this research and development activity and providing valuable feedback.

REFERENCES

[1] "BUILDER." Retrieved from https://builder.nrl.navy.mil, September 2010.
[2] A. S. Yilmaz, B. N. Mcquay, H. Yu, A. S. Wu, and J. C. Sciortino, "Evolving sensor suites for enemy radar detection," in *Genetic and Evolutionary Computation GECCO*, 2003.
[3] T. Shima and C. Schumacher, "Assigning cooperating uavs to simultaneous tasks on consecutive targets using genetic algorithms," *Journal of the Operational Research Society*, vol. 60, no. 7, 2009.
[4] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. Morgan Kaufmann, 2001.
[5] H. Liu, B. Li, Y. Ji, and T. Sun, *Applied Soft Computing Technologies: The Challenge of Complexity*, ch. Particle Swarm Optimization: from lbest to gbest. Heidelberg: Springer-Berlin, 2006.
[6] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. Wiley, 2007.
[7] P. Suganthan, "Particle swarm optimiser with neighborhood operator," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1958–1962, 1999.
[8] K. E. Parsopoulos, D. K. Tasoulis, M. N. Vrahatis, and K. Words, "Multiobjective optimization using parallel vector evaluated particle swarm optimization," in *In Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA 2004*, pp. 823–828, ACTA Press, 2004.
[9] M. Abramson and R. Mittu, *Challenges in Large-Scale Coordination*, ch. Multiagent Coordination in Open Environments. Springer, 2005.
[10] S. K. Smit and A. E. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, 2009.
[11] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, 1986.
[12] V. Nannen and A. E. Eiben, "Relevance estimation and value calibration of evolutionary algorithm parameters," in *International Joint Conference of Artificial Intelligence IJCAI*, 2007.
[13] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artificial Intelligence Review*, vol. 11, pp. 193–225, 1997.
[14] T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuss, "Sequential parameter optimization," in *IEEE Congress on Evolutionary Computation*, pp. 773–780, 2005.
[15] T. Bartz-Beielstein and S. Markon, "Tuning search algorithms for real-world applications: A regression tree based approach," in *IEEE Congress on Evolutionary Computation*, pp. 1111–1118, 2004.
[16] G. E. Box and N. R. Draper, *Empirical Model-building and response surfaces*. John Wiley and Sons, Inc., 1987.
[17] J. Albus, *Brains, Behaviour, and Robotics*. McGraw-Hill, 1981.
[18] W. Miller, F. Glanz, and L. Kraft, "CMAC: an associative neural network alternative to backpropagation," in *Proceedings of the IEEE*, pp. 1561–1567, 1990.
[19] R. Sutton, "Generalization in reinforcement learning: successful examples using sparse coarse coding," *Advances in Neural Information Processing Systems*, vol. 8, pp. 1038–1044, 1996.
[20] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communication of the Association of Computing Machinery CACM*, vol. 13, pp. 422–426, 1970.
[21] V. Vapnik, *Estimation of dependencies based on empirical data*. Springer-Verlag, 2006.
[22] S. lin and R. Wright, "Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning," in *Workshop at the Twenty-fourth AAAI Conference on Artificial Intelligence*, 2010.
[23] "WEKA." Retrieved from http://www.cs.waikato.ac.nz/ml/weka/, December 2009.